IN THE CLAIMS:

1. (Currently Amended) A system comprising:

a shared resource;

multiple processors arranged to access said shared resource; and

an operating system configured to allow said multiple processors to perform work on said shared resource concurrently while supporting state changes or updates of said shared resources, said operating system comprising a synchronization algorithm for synchronizing multiple threads of operation with a single thread so as to achieve mutual exclusion between multiple threads performing work on said shared resource and a single thread updating or changing the state of said shared resource without requiring serialization of all threads such that an update or change of the state of the shared resource may be made by the single thread only when none of the multiple threads are processing work on the shared resource.

2. (Original) The system as claimed in claim 1, wherein said shared resource includes work queues associated with a hardware adapter configured to send and receive message data to/from a remote system.

3. (Original) The system as claimed in claim 2, wherein said synchronization algorithm is executed to synchronize any thread wishing to update or change a state of said shared resource with all the threads processing I/O operations on said shared resource.

4. (Original) The system as claimed in claim 1, wherein said synchronization algorithm is executed to allow worker threads to work concurrently while processing I/O operations in

exclusion of an update thread when a state of said shared resource is not changing, and allow an update thread to change the state or update said shared resource in exclusion of multiple worker threads.

5. (Original) The system as claimed in claim 4, wherein said synchronization algorithm is executed to support a worker thread operation for processing simultaneous I/O operations on said shared resource while concurrently supporting an update thread operation for updating or changing the state of said shared resource.

6. (Original) A system as claimed in claim 5, wherein said worker thread operation is invoked by one of an event and a user's request, and is performed by:

determining whether a lock is available;

if the lock is not available, waiting until the lock becomes available;

if the lock is available, seizing the lock while incrementing a count by a discrete constant to indicate the number of worker threads that are active, and then releasing the lock after the count has been incremented;

after the lock has been released, allowing multiple worker threads to process work concurrently;

determining next whether there is work to be processed;

if there is work to be processed, processing the work until there is no work to be processed; and

if there is no work to be processed, decrementing the count by a discrete constant to indicate when all the worker threads are done with completion processing.

7. (Original) A system as claimed in claim 6, wherein said update thread operation is invoked by a user's request, and is performed by:

determining whether a lock is available;

if the lock is not available, waiting until the lock becomes available when released by any one of the worker threads;

if the lock is available, seizing the lock until the count becomes zero (0) to indicate that it is safe to update or change the state of said shared resource, and updating or changing the state of said shared resource; and

after said shared resource has been updated, releasing the lock so as to allow either new worker threads to continue I/O operation processing or a different update thread to continue shared resource updating.

8. (Original) A system as claimed in claim 2, further comprising data channels formed between said system and said remote system, via a switched fabric, and supported by the "*Virtual Interface (VI) Architecture Specification*" and the *"Next Generation Input/Output (NGIO) Specification*" for message data transfers between said system and said remote system.

9. (Original) The system as claimed in claim 2, wherein said synchronization algorithm is installed as part of a software driver module of an operating system (OS) kernel or an user-level application of said system.

10. (Original) The system as claimed in claim 2, wherein said shared resource includes ones of work queues, completion queues, FIFO queues, hardware adapters, I/O controllers and other memory elements of said system.

11. (Currently Amended) A network, comprising:

a switched fabric;

remote systems attached to said switched fabric; and

a host system comprising multiple processors; a host-fabric adapter provided to interface with said switched fabric and included work queues each configured to send and receive message data from a single remote system, via said switched fabric; and an operating system configured to allow said multiple processors to perform work on said work queues concurrently while supporting state changes of said work queues, said operating system comprising a synchronization algorithm for synchronizing multiple threads of operation with a single thread so as to achieve mutual exclusion between multiple threads performing work on said work queues and a single thread changing the state of said work queues without requiring serialization of all threads such that an update or change of the state of the work queues may be made by the single thread only when none of the multiple threads are processing work on the work queues.

12. (Original) The network as claimed in claim 11, wherein said synchronization algorithm is executed to synchronize any thread wishing to update or change a state of said work queues with all the threads processing I/O operations on said work queues.

13. (Original) The network as claimed in claim 11, wherein said synchronization algorithm is executed to allow worker threads to work concurrently while processing I/O operations in exclusion of an update thread when the state of said work queues is not changing, and allow an update thread to change the state or update said work queues in exclusion of multiple worker threads.

14. (Original) The network as claimed in claim 11, wherein said synchronization algorithm is executed to support a worker thread operation for processing simultaneous I/O operations on said work queues while concurrently supporting an update thread operation for updating or changing the state of said work queues.

15. (Original) A network as claimed in claim 14, wherein said worker thread operation is invoked by one of an event and a user's request, and is performed by:

determining whether a lock is available;

if the lock is not available, waiting until the lock becomes available;

if the lock is available, seizing the lock while incrementing a count by a discrete constant to indicate the number of worker threads that are active, and then releasing the lock after the count has been incremented;

after the lock has been released, allowing multiple worker threads to process work concurrently;

determining next whether there is work to be processed;

if there is work to be processed, processing the work until there is no work to be processed; and

if there is no work to be processed, decrementing the count by a discrete constant to indicate when all the worker threads are done with completion processing.

16. (Original) A network as claimed in claim 14, wherein said update thread operation is invoked by a user's request, and is performed by:

determining whether a lock is available;

if the lock is not available, waiting until the lock becomes available when released by any one of the worker threads without any work to be processed;

if the lock is available, seizing the lock until the count becomes zero (0) to indicate that it is safe to update or change the state of said shared resource, and updating or changing the state of said work queues; and

after said work queues have been updated, releasing the lock so as to allow either new worker threads to continue I/O operation processing or a different update thread to continue work queue updating.

17. (Original) A network as claimed in claim 11, further comprising data channels formed between said host system and said remote systems, via said switched fabric, and supported by the "*Virtual Interface (VI) Architecture Specification*" and the *"Next Generation Input/Output (NGIO) Specification*" for message data transfers between said host system and said remote systems.

18. (Original) The network as claimed in claim 11, wherein said synchronization algorithm is installed as part of a software driver module of an operating system (OS) kernel or an user-level application of said host system.

19. (Original) The network as claimed in claim 11, wherein said host system and said remote systems represent channel endpoints of a data network implemented in compliance with the *"Next Generation Input/Output (NGIO) Specification"*, and data channels formed between said host system and said remote systems, via said switched fabric, are supported by the "*Virtual Interface (VI) Architecture Specification*" and the *"Next Generation Input/Output (NGIO) Specification"* for message data transfers between said host system and said remote systems.

20. (Original) A process of synchronizing an update thread which updates a list of work queues with multiple worker threads which operate on items in the list of work queues in a multi-processor system, comprising:

allowing a group of worker threads to concurrently access the list of work queues to process I/O operations in exclusion of an update thread, when states of the work queues are not changing;

incrementing a count of threads processing I/O operations each time a worker thread is running, while decrementing the count of threads processing I/O operations each time a worker thread is done processing I/O operations;

when the count of threads reaches a designated value indicating that no worker threads are running, allowing an update thread to access and update the list of work queues in exclusion of new worker threads from processing I/O operations; and

after the list of work queues is updated, allowing new worker threads to perform I/O operations until all worker threads are done processing I/O operations.

21. (Original) A computer-readable medium that stores computer-executable instructions for synchronizing an update thread which updates a list of work queues with a group of threads which operate on items in the list of work queues in a multi-processor system, said computer-executable instructions causing said multi-processor system to:

permit a group of worker threads to concurrently access the list of work queues to process I/O operations in exclusion of an update thread, when states of the work queues are not changing;

increment a count of threads processing I/O operations each time a worker thread is running, while decrementing the count of threads processing I/O operations each time a worker thread is done processing I/O operations;

permit a single update thread to access and update the list of work queues in exclusion of new worker threads from processing I/O operations, when the count of threads reaches a designated value indicating that no worker threads are running; and

after the list of work queues is updated, permit new worker threads to perform I/O operations until all worker threads are done processing I/O operations.